Anexa 2 - Shell scripting

Contents

Shell scripting

Shell-ul este principala interfață de comunicare între utilizator şi sistemul de operare. Deşi, în mod intuitiv, shell-ul este identificat cu o interfață în linia de comandă, poate fi şi o interfață grafică. Exemplu este Explorer-ul sistemului de operare Windows.

În cele ce urmează ne vom oferi la interfața de tip CLI (Command Line Interface) oferită de sistemele de operare Unix. Deşi cu o curbă de învățare mai mare decat o interfață grafică, CLI permite un control mult mai bun al sistemului. Mai mult, shell-ul dispune de un limbaj de programare. Un program shell, denumit script shell, este folosit pentru a îmbina mai multe comenzi şi diverse structuri de control pentru a obține o nouă funcționalitate sau pentru automatizarea sarcinilor. În acest fel un script shell este un instrument esențial pentru sarcinile administrative şi alte rutine repetitive care nu necesită funcționalități ale unor limbaje de programare avansate.

În continuare ne vom referi la Bash (Bourne Again SHell). Există şi alte shell-uri pe sisteme Unix precum tcsh, zsh, ash, etc. De curând, Microsoft oferă PowerShell pe sistemele Windows. PowerShell are o abordarea orientată pe obiecte şi un set de funcționalități care acoperă nevoile de administrare ale unui sistem Windows.

Cel mai simplu script shell

Un script simplu care doar afişează mesajul `"Hello, World!"` este următorul:

```
#!/bin/bash

# afiseaza mesaj
echo "Hello, World!"

exit 0
```

Execuția acestuia este următoarea:

```
razvan@asgard:~/school/2006/so/shell$ chmod +x hello.sh
razvan@asgard:~/school/2006/so/shell$ ./hello.sh
Hello, World!
```

Se observă că este necesar ca fişierul să fie executabil pentru a putea fi interpretat. Şirul **#!** de la începutul fişierului poartă denumirea de *shebang*. Acesta indică sistemului ce program va fi invocat pentru a interpreta scriptul. Exemple pot fi:

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/python
#!/usr/bin/awk -f
```

Spre exemplu, următorul script se şterge pe sine:

```
#!/bin/rm -f

aici putem scrie orice ... oricum se va şterge
```

Un script poate fi rulat prin precizarea explicită a interpretorului în linia de comandă:

```
razvan@asgard:~/school/2006/so/shell$ bash hello.sh
Hello, World!
```

În această situație nu este nevoie ca scriptul sa fie executabil şi nici nu este nevoie de prezența liniei **#!/bin/bash**.

Caracterul **#** semnifică începutul unui comentariu care durează pană la sfarşitul liniei.

Comanda **exit** este folosită pentru a indica valoarea de retur a scriptului. Este implicit 0 (cu alte cuvinte nu era necesar să apară în script).

Operatori shell

Shell-ul prezinta o serie de operatori folositi pentru imbinarea comenzilor.

Concatenarea comenzilor

Următorii operatori sunt folositi pentru concatenarea diverselor comenzi:

- **command1 ; command2** - comenzile sunt executate secvențial (una dupa alta)
- **command1 && command2** - command2 este executată numai dacă `command1` are valoare de retur 0
- **command1 || comand2** - command2 este executată numai dacă `command1` are valoare de retur diferita de 0

Inlantuirea comenzilor

Inlantuirea comenzilor se realizeaza folosind operatorul | (pipe). In aceasta situatie iesirea unei comenzi devine intrarea pentru cealalta.

Cateva exemple sunt prezentate in continuare:

```
$ last -30 | grep Tue
razvan   pts/2        :0.0             Tue Jan  2 20:42 - down   (05:12)
razvan   pts/2        :0.0             Tue Jan  2 20:35 - 20:41  (00:06)
razvan   pts/1        :0.0             Tue Jan  2 20:34 - 21:23  (00:48)
razvan   pts/0        :0.0             Tue Jan  2 20:27 - down   (05:27)
wtmp begins Tue Nov 14 04:22:33 2006
$ last -30 | grep Tue | tr -s ' '
razvan pts/2 :0.0 Tue Jan 2 20:42 - down (05:12)
razvan pts/2 :0.0 Tue Jan 2 20:35 - 20:41 (00:06)
razvan pts/1 :0.0 Tue Jan 2 20:34 - 21:23 (00:48)
razvan pts/0 :0.0 Tue Jan 2 20:27 - down (05:27)
wtmp begins Tue Nov 14 04:22:33 2006
$ last -30 | grep Tue | tr -s ' ' | head -4
razvan pts/2 :0.0 Tue Jan 2 20:42 - down (05:12)
razvan pts/2 :0.0 Tue Jan 2 20:35 - 20:41 (00:06)
```

```
razvan pts/1 :0.0 Tue Jan 2 20:34 - 21:23 (00:48)
razvan pts/0 :0.0 Tue Jan 2 20:27 - down (05:27)
$ last -30 | grep Tue | tr -s ' ' | head -4 | cut -d ' ' -f 2
pts/2
pts/2
pts/1
pts/0
$ last -30 | grep Tue | tr -s ' ' | head -4 | cut -d ' ' -f 2 | uniq
pts/2
pts/1
pts/0
$ last -30 | grep Tue | tr -s ' ' | head -4 | cut -d ' ' -f 2 | uniq | wc -l
3
```

Redirectari

Comenzilor le pot fi redirectate, respectiv, intrarea standard, iesirea standard si eroarea standard dintr-un fisier. O parte din operatorii folositi pentru redirectare sunt:

- **>** - redirectarea iesirii standard
- **2>** - redirectarea erorii standard
- **2>&1** - redirectarea erorii standard in iesirea standard. Efectiv, unificarea stderr cu stdout.
- **<** - redirectarea intrarii standard

Exemple:

```
$ ls -l > ls.out
$ strace ls 2> strace.out
$ grep "alpha" < file.txt
```

Variabile

Ca orice limbaj, shell-ul permite utilizarea de variabile. Spre deosebire de limbajele cunoscute, variabilele shell nu au tipuri. O variabila poate fi evaluata atat ca numar cat si ca sir de caractere.

Exemple de definire de variabile:

```
var1=2
var2=4asa
var3='abcd'
my_var="asdf"
my_other_var="a 1 3 4"
new_var=$var1
new_var2=${var2}var3
```

ATENTIE! Sintaxa shell este foarte stricta; **NU** este permis sa existe spatii intre numele variabilei si caracterul **=** sau intre caracterul **=** si valoarea variabilei.

Se observa ca valoarea unei variabile este referita prin folosirea simbolului **$**.

Exemple de folosire de variabile:

```
razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo $var1
2
razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo $var12

razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo ${var1}2
22
razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo "$var1"
2
razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo "$var1"2
22
razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo $var1$my_other_var
2a 1 3 4
razvan@anaconda:~/mystuff/cfiles/solab/labs$ echo "$var1 $my_other_var"
2 a 1 3 4
```

Argumente in linia de comanda

Un script poate primi argumente in linia de comanda. Argumentele sunt referite respectiv folosind parametrii pozitionali: **$1**, **$2**, ... **$0** este numele scriptului (echivalent cu argv[0] din C).

Numarul de argumente din linia de comanda este dat de **$#**. **$#** va fi 0 daca nu avem argumente in linia de comanda (echivalentul C - argc - ar fi avut valoarea 1 in acest caz).

$@ poate fi folosit pentru obtinerea intregii liste de argumente separate prin spatiu.

Exemplu:

```
#!/bin/sh

echo "Scriptul are $# parametri"
echo "Numele scriptului este $0"

if test $# -ge 1; then
        echo "Primul parametru este $1"
fi

if test $# -ge 2; then
        echo "Cel de-al doilea parametru este $2"
fi

echo "Lista de parametri este $@"
```

Rularea scriptului este:

```
razvan@asgard:~/test/so$ chmod +x test.bash
razvan@asgard:~/test/so$ ./test.bash alfa "beta gamma"
Scriptul are 2 parametri
Numele scriptului este ./test.bash
Primul parametru este alfa
Cel de-al doilea parametru este beta gamma
Lista de parametri este alfa beta gamma
```

shift

Comanda builtin **shift** este folosita pentru deplasarea parametrilor pozitionali cu valoarea primita ca parametru (sau 1 daca nu este prezenta). Astfel daca se primeste valoarea N, parametrii pozitionali de la N+1 la **$#** vor fi redumiti la **$1**, **$2**, ... **$#-N+1**

Exemplu:

```
razvan@asgard:~/test/so$ cat pos2.sh
#!/bin/sh

if test $# -ge 2; then
        echo "Parametrii inainte de shift sunt $@"
        shift
        echo "Parametrii dupa shift sunt $@"
fi
razvan@asgard:~/test/sp$ ./pos2.sh a b c
Parametrii inainte de shift sunt a b c
Parametrii dupa shift sunt b c
```

Caractere speciale

Un set de caractere au semnificatie speciala in cadrului shell-ului.

spatiu

Caracterul **spatiu** (blank) este separator pentru argumentele in linia de comanda sau pentru elementele unei liste. Daca se doreste transmiterea ca parametru a unui argument ce contine spatiu acesta trebuie citat (quoted):

```
$ ls my\ dir
$ ls "my dir"
$ ls 'my dir'
```

backslash

Caracterul **backslash** forteaza caracterul ce-l precede sa-si pastreze semnificatia literala; cu alte cuvinte, este folosit pentru a intarzia (a escapa) acel caracter:

```
razvan@asgard:~/school/2006/so/shell$ echo $var1
test
razvan@asgard:~/school/2006/so/shell$ echo \$var1
$var1
```

ghilimele

Un sir intre **ghilimele** (double quotes ") va pastra semnificatia literala a caracterelor ce-l compun cu exceptia caracterelor ' (apostrof) si **$** (dollar). Caracterul **** (backslash) isi pastreaza semnificatia speciala numai in cazul in care este urmat de $, ', ``, \ sau newline.

apostrof

Un sir intre caractere **apostrof** (single quotes) va pastra semnificatia literala a **tuturor** caracterelor ce-l compun (nu exista exceptii).

dollar - expansion

Caracterul **dollar** este folosit in mai multe situatii in ceea ce se numeste **expansion**. Este folosit pentru a recupera valoarea unei variabile, pentru a stoca intr-o variabila iesirea unei functii, etc.

expandarea unui parametru

Se inlocuieste un parametru (variabila) sau se inlocuiesc parametrii pozitionali. **$?** se traduce in valoarea de retur a ultimei comenzi executate.

```
razvan@asgard:~/school/2006/so/shell$ ls -l | grep 126
-rwxr-xr-x  1 razvan razvan   126 2007-01-06 21:42 pos2.sh
razvan@asgard:~/school/2006/so/shell$ echo $?
0
razvan@asgard:~/school/2006/so/shell$ ls -l | grep 1267
razvan@asgard:~/school/2006/so/shell$ echo $?
1
```

substitutia unei comenzi

Iesirea unei comenzi inlocuieste comanda efectiva:

```
razvan@asgard:~/school/2006/so/shell$ ls -l | wc -l
11
razvan@asgard:~/school/2006/so/shell$ var=$(ls -l | wc -l)
razvan@asgard:~/school/2006/so/shell$ echo $var
11
```

expansiune aritmetica

Se realizeaza evaluarea unei expresii aritmetice cu furnizarea rezultatului:

```
razvan@asgard:~/school/2006/so/shell$ num="1+2*3"
razvan@asgard:~/school/2006/so/shell$ echo $num
1+2*3
razvan@asgard:~/school/2006/so/shell$ num=$((1+2*3))
razvan@asgard:~/school/2006/so/shell$ echo $num
7
```

Structuri de control

Ca orice limbaj de programare, shell-ul are un set de structuri de control pentru a permite lucrul cu cicluri si cu decizii.

if

Sintaxa pentru **if** este urmatoarea:

```
if CONDITIE1; then
        comenzi
elif CONDITIE2; then
        comenzi
else
        comenzi
fi
```

Structurile **elif** si **else** sunt optionale.

Conditia poate aparea in doua formate:

```
[ conditie_efectiva ]
```

sau

```
test conditie_efectiva
```

Pentru conditiile posibile consultati pagina de manual a utilitarului **test**.

ATENTIE! Daca folositi prima varianta este nevoie de spatiu dupa **[** si inainte de **]**.

Exemple:

```
 if test $i -eq 3; then
        echo "valoarea este 3"
 else
        echo "valoarea este diferita de 3"
 fi

 if test "$var" = "alfa" -o "$var" = "beta"; then
        echo "valoarea este alfa sau beta"
 fi

 if test -f "$i"; then
        echo "$i este un fisier"
 elif test -d "$i"; then
        echo "$i este un director"
 else
        echo "$i nu este nici fisier nici director"
 fi
```

case

Sintaxa pentru **case** este urmatoarea:

```
case VARIABILA in
        pattern1) comenzi ;;
        pattern2) comenzi ;;
esac
```

Se poate folosi ca pattern * pe post de default.

Exemplu:

```
 #!/bin/sh

 echo -n "Enter the name of an animal: "
 read ANIMAL

 echo -n "The $ANIMAL has "
 case $ANIMAL in
        horse | dog | cat) echo -n "four";;
        man | kangaroo ) echo -n "two";;
        *) echo -n "an unknown number of";;
 esac
 echo " legs."
```

for

Sintaxa pentru **for** este urmatoarea:

```
for VARIABILA in LISTA; do
        comenzi
done
```

LISTA este o insiruire de elemente separate prin spatii. Variabila va lua pe rand aceste valori. Daca se doresc variabile numerice in stilul C se poate folosi constructia ((**...**)).

Exemple:

```
 for i in *; do
        if test -f "$i"; then
                echo "$i este fisier"
        fi
 done

 for i in "1 2 3"; do
        sum=$(($sum + $i))
 done
 echo "suma este $sum"

 for i in $(seq 1 10); do
        sum=$(($sum + $i))
 done
 echo "suma primelor 10 numere este $sum"
```

```
for ((i = 0; i < 100; i += 2)); do
        sum=$(($sum + $i))
done
echo "suma numerelor pare pana la 100 este $sum"
```

while

Sintaxa pentru while este urmatoarea:

```
while CONDITIE; do
        comenzi
done
```

Conditia are acelasi format ca la **if**.

Exemplu:

```
i=1
prod=1
while test $i -le 30; do
        prod=$(($prod * $i))
        i=$(($i + 5))
done
echo "produsul este $prod"
```

Functii shell

Ca orice alt limbaj de programare, shell-ul permite lucrul cu functii proceduri. Sintaxa unei functii este:

```
[function] nume_functie ()
{
        comenzi
}
```

Identificatorul **function** este optional. Sintaxa de apel este simpla: numele functiei urmat de eventualii parametri.

ATENTIE! In cadrul unei functii argumentele vor fi referite tot ca parametri pozitionali (**$1**, **$2**, ...) astfel incat daca dorim sa referim intr-o functie argumentele in linia de comanda va trebui sa-i transmitem ca parametri la apelul functiei.

Exemplu:

```
#!/bin/sh

function fun1 ()
{
        echo "Aceasta este functia fun1"
}

function fun2 ()
{
        echo "Aceasta este functia fun2"
```

```
        echo "Argumentele functiei sunt $1 $2"
}

fun1
fun2 a b
fun2 alfa
```

Pattern matching

In interactiunea cu sistemul de fisiere se doreste selectarea rapida a mai multor fisiere dupa cateva caracteristici de nume comune. Operatia efectuata de shell se numeste pattern matching. Exista urmatoarele caractere speciale:

- ***** - se potriveste cu orice sir de caracter, inclusiv sirul vid
- **?** - se potriveste cu un singur caracter
- **[...]** - se potriveste cu unul din caracterele din set; poate fi de genul [abc] sau [a-gA-G] sau [0-5h-zH-Z]

Pentru urmatoarele pattern-uri trebuie activata optiunea shopt -s extglob.

- **{sir1,sir2,sir3,..}** - se potriveste cu unul dintre sirurile dintre acolade
- **?(lista_sabloane)** - se potriveste cu o aparitie sau nici una a unuia dintre sabloane
- ***(lista_sabloane)** - se potriveste cu nici o aparitie sau mai multe a unuia dintre multe sabloane
- **+(lista_sabloane)** - se potriveste cu o aparitie sau mai multe a unuia dintre sabloane
- **@(lista_sabloane)** - se potriveste cu exact un sablon din lista
- **!(lista_sabloane)** - se potriveste cu toate sabloanele din lista mai putin unul

Exemplu:

```
razvan@asgard:~/school/2006/so/shell$ ls *.sh
fun.sh  hello.sh  pos2.sh  pos.sh
razvan@asgard:~/school/2006/so/shell$ ls ?h*
shell2.lyx  shell.lyx  shell.tex
razvan@asgard:~/school/2006/so/shell$ ls *{e,y}x
lab8.lyx  shell2.lyx  shell.lyx  shell.tex
razvan@asgard:~/school/2006/so/shell$ ls *[a-h]e*
hello.sh  shell2.lyx  shell.lyx  shell.tex
razvan@asgard:~/school/2006/so/shell$
```

Comenzi utile

O serie de comenzi (interne sau externe) sunt utile în crearea de scripturi shell.

echo

Comanda `echo` este folosită pentru a afişa un şir de caractere, o variabilă la ieşire standard:

```
razvan@asgard:~/test/so$ echo alfa
alfa
```

```
razvan@asgard:~/test/so$ echo $PATH
/home/razvan/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
razvan@asgard:~/test/so$ echo '$PATH'
$PATH
```

cat, tac, head, tail

Comanda `cat` afişează conținutul unui fişier sau al intrării standard. Comanda `tac` afişează conținutul unui fişier inversat. Comanda `head` afişează începutul unui fişier sau al intrării standard, în timp ce comanda `tail` afişează sfârşitul acestuia:

```
razvan@asgard:~/test/so$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
[...]
razvan@asgard:~/test/so$ tac /etc/passwd
bogdan:x:1006:1005:,,,:/home/students/bogdan:/bin/bash
monica:x:1005:1005:,,,:/home/students/monica:/bin/bash
lucian:x:1004:1004:,,,:/home/lucian:/bin/bash
[...]
razvan@asgard:~/test/so$ head -n 3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
razvan@asgard:~/test/so$ tail -n -3 /etc/passwd
lucian:x:1004:1004:,,,:/home/lucian:/bin/bash
monica:x:1005:1005:,,,:/home/students/monica:/bin/bash
bogdan:x:1006:1005:,,,:/home/students/bogdan:/bin/bash
```

read

Comanda `read` este folosită pentru citirea de informații de la intrarea standard:

```
razvan@asgard:~/test/so$ read a
alfa
razvan@asgard:~/test/so$ echo $a
alfa
```

O folosirea frecventă este în combinație cu `while` pentru citirea linie cu linie a conținutului unui fişier:

```
razvan@asgard:~/test/so$ cat out.txt | while read a; do echo "am citit $a"; done
am citit a
am citit b
am citit c
```

find

Comanda `find` este o comandă fundamentală pentru parcurgerea intrărilor dintr-o ierarhie a sistemului de fişiere. Câteva exemple de utilizare sunt prezentate în continuare:

- cautarea headerelor de sistem care încep cu şirul `std`:

```
razvan@asgard:~/test/so$ find /usr/include/ -type f -name 'std*'
/usr/include/c++/4.1.2/ext/stdio_sync_filebuf.h
/usr/include/c++/4.1.2/ext/stdio_filebuf.h
/usr/include/c++/4.1.2/stdexcept
/usr/include/stdio.h
/usr/include/linux/stddef.h
/usr/include/stdlib.h
/usr/include/bits/stdio-lock.h
/usr/include/bits/stdio.h
/usr/include/bits/stdio2.h
/usr/include/bits/stdio_lim.h
/usr/include/stdint.h
/usr/include/nptl/bits/stdio-lock.h
/usr/include/stdio_ext.h
```

- căutarea şirului `mutex` în headerele de sistem care conţin şirul `lock`

```
razvan@asgard:~/test/so$ find /usr/include/ -name '*lock*' -exec grep -H mutex {} \;
/usr/include/linux/seqlock.h: * This can be used when code [...]
/usr/include/linux/seqlock.h: * own mutexing.
/usr/include/linux/lockdep.h: * rwlocks, mutexes and rwsems) [...]
/usr/include/linux/lockdep.h:#  define mutex_acquire(l, s, t, i)  [...]
```

Opţiunea `exec` este folosită pentru a rula o comandă pentru fişierele găsite. Şirul {} este special şi se înlocuieşte cu numele fişierului găsit de `find`. Caracterul `;` (citat cu ajutorul backslash) indică încheierea comenzii de executat.

Filtre de text

Variabilele, structurile de control si procedurile sunt intalnite in toate limbajele de programare. Ce face insa un script shell indicat pentru sarcini administrative si repetitive este posibilitatea de imbinare a comenzilor simple, de lucru cu fisierele sistemului pentru a obtine informatiile dorite si pentru a adauga o noua functionalitate. De obicei aceste sarcini necesita o procesare sofisticata. In aceste situatii se folosesc filtrele de text.

Comenzi folosite ca si filtre de text sunt **head**, **tail**, **grep**, **sort**, **uniq**, **tr**, **cut**.

head, tail

Cele doua comenzi sunt folosite pentru afisarea numai a primelor sau a ultimelor linii de text din cadrul unui fisier. Sintaxa lor este asemanatoare:

```
head [-n lines] files
tail [-n lines] files
```

Primul argument, daca exista, afiseaza primele, respectiv ultimele n linii din text. Lipsa acestuia impune n = 10.

Exemple de comenzi sunt:

```
razvan@ragnarok:~/cfiles/solab/labs$ ls -l
total 44
```

```
drwxr-xr-x  2 razvan razvan 4096 Nov 16 00:01 lab1
drwxr-xr-x  3 razvan razvan 4096 Oct 12 00:02 lab2
-rwxr-xr-x  1 razvan razvan  937 Oct 19 00:53 lab2html.sh
drwxr-xr-x  3 razvan razvan 4096 Oct 19 00:50 lab3
drwxr-xr-x  3 razvan razvan 4096 Oct 26 01:09 lab4
drwxr-xr-x  3 razvan razvan 4096 Nov  2 09:47 lab5
drwxr-xr-x  3 razvan razvan 4096 Nov 12 19:56 lab6
drwxr-xr-x  2 razvan razvan 4096 Nov 16 12:43 lab7
drwxr-xr-x  2 razvan razvan 4096 Nov 20 12:34 lab8
drwxr-xr-x  5 razvan razvan 4096 Oct 17 19:52 solab
drwxr-xr-x  2 razvan razvan 4096 Nov 12 20:06 tema1
razvan@ragnarok:~/cfiles/solab/labs$ ls -l | head -3
total 44
drwxr-xr-x  2 razvan razvan 4096 Nov 16 00:01 lab1
drwxr-xr-x  3 razvan razvan 4096 Oct 12 00:02 lab2
razvan@ragnarok:~/cfiles/solab/labs$ ls -l | tail -4
drwxr-xr-x  2 razvan razvan 4096 Nov 16 12:43 lab7
drwxr-xr-x  2 razvan razvan 4096 Nov 20 12:34 lab8
drwxr-xr-x  5 razvan razvan 4096 Oct 17 19:52 solab
drwxr-xr-x  2 razvan razvan 4096 Nov 12 20:06 tema1
razvan@ragnarok:~/cfiles/solab/labs$
```

In cazul comenzi **tail** o optiune utila este **-f** care mentine fisierul de vizualizat deschis pe masura ce programele scriu in el, spre exemplu pentru a vizualiza un fisier de log pe masura ce informatiile sunt scrise in el:

```
$ tail -f /var/log/apache/access_log
```

grep

Comanda grep permite localizarea liniilor intr-un fisier care contine o expresie cautata. Sintaxa de baza este

```
$ grep word file
```

file este numele fisierului in care vrem sa gasim cuvantul **word**. Un exemplu de utilizare este:

```
razvan@ragnarok:~/cfiles/solab/labs$ grep latex lab2html.sh
latex2html "$BASE.tex" 2>&1 > /dev/null
latex "$BASE.tex" 2>&1 > /dev/null
```

De multe ori dorim sa realizam cautarea in mod case-insensitive (adica sa nu conteze faptul ca se cauta UNIX sau unix). Pentru aceasta folosim optiunea **-i**.

Cand nu se precizeaza un fisier se foloseste intrarea standard, **grep** devenind ideal pentru lucrul cu pipes. De exemplu:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ last | head -100 | grep tty
root     tty1                         Wed Nov 16 11:00 - down   (00:00)
razvan   tty1                         Wed Nov 16 00:07 - 00:08  (00:01)
```

O alta optiune utila este **-v**. Aceasta permite cautarea acelor linii care NU contin cuvantul transmis ca parametru. Spre exemplu, daca dorim afisarea utilizatorilor care nu au ca si director de baza un director de forma **/home/nume**, vom folosi comanda:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ grep -v /home /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
.......
```

Un alt exemplu de folosire este parsarea output-ului comenzii ps:

```
razvan@ragnarok:~/cfiles/solab/labs/lab8$ ps -af | grep lyx
razvan    3767  3719  0 12:19 pts/0    00:00:05 lyx
razvan    4219  3719  0 12:53 pts/0    00:00:00 grep lyx
razvan@ragnarok:~/cfiles/solab/labs/lab8$ ps -af | grep lyx | grep -v grep
razvan    3767  3719  0 12:19 pts/0    00:00:05 lyx
```

Optiunea **-n** permite afisarea numarului liniei care continea cuvantul cautat.

Putem de asemenea sa afisam numai fisierele care contin acel cuvant (fara afisarea liniilor). Pentru aceasta folosim optiunea **-l** (listare). De obicei este folosita in conjunctie cu optiunea **-R** pentru cautarea recursiva in cadrul unei structuri de directoare:

```
razvan@ragnarok:~/cfiles/solab/labs$ grep -r -l "#include <sys/wait.h>" .
./lab2/lab2.lyx
./lab2/lab2.tex
./lab2/wait_zombie.c
./lab2/simple_exec.c
./solab/lab2/lab2.lyx
./solab/lab2/lab2.tex
./solab/lab2/wait_zombie.c
./solab/lab2/simple_exec.c
./lab6/lab6.lyx
./lab6/mmap.c
./lab6/lab6.tex
```

tr

Comanda **tr** (transliterate) este folosita pentru a translata caracterele dintr-un set de caractere intr-un alt set de caractere. Sintaxa de baza este:

```
tr 'set1' 'set2'
```

Spre exemplu, daca am dori sa aflam numarul de cuvinte dintr-un text, am translata toate caracterele speciale in spatii cu o comanda de forma:

```
$ tr '!?":'\[\]{}(),.' ' ' < file
```

Caracterele **[** si **]** au fost escapate folosind ****. Daca dorim sa translatam literele mari in litere mici, folosim:

```
$ tr 'A-Z' 'a-z' < file
```

In cazul in care setul **set2** are mai putine caractere decat setul **set1** acestea vor fi multiplicate pentru a ajunge la aceeasi dimensiune.

Urmatorul pas ar fi eliminarea spatiilor redundante. Optiunea **-s** (squeeze) inlocuieste o succesiune de doua sau mai multe caractere cu unul singur. Un exemplu este:

```
razvan@ragnarok:~/cfiles/solab/labs$ echo shell programming | tr -s 'lm'
shel programing
```

sort

Comanda sort este utilizata pentru sortarea liniilor alfabetic. Un exemplu de utilizare este:

```
razvan@ragnarok:~/cfiles/solab/labs$ echo -en "alfa\nomega\ngamma\nbeta\n"
alfa
omega
gamma
beta
razvan@ragnarok:~/cfiles/solab/labs$ echo -en "alfa\nomega\ngamma\nbeta\n" | sort
alfa
beta
gamma
omega
```

O optiune utila in cazul sort este sortarea dupa valoarea numerica a sirurilor. Pentru aceasta folosim **-n**. De exemplu:

```
razvan@ragnarok:~/cfiles/solab/labs$ echo -en "10\n43\n4\n9\n123\n5\n" | sort 10
123
4
43
5
9
razvan@ragnarok:~/cfiles/solab/labs$ echo -en "10\n43\n4\n9\n123\n5\n" | sort -n
4
5
9
10
43
123
```

De multe ori output-ul apare intr-o forma in care elementele de sortat sunt intr-o alta coloana (nu prima, cea folosita implicit de sort). Pentru aceasta se poate folosi optiunea **-k** (key). Sintaxa, in cazul folosirii acestei optiuni, este:

```
sort -k start, end file
```

Aici **start** este coloana de start a cheii, iar **end** este coloana de stop a cheii. Prima coloana este cea dupa care se face sortarea, iar, in cazul in care sunt doua sau mai multe elemente egale, se face deosebirea intre acestea folosind coloana urmatoare din cheie, etc.

Exemplu de utilizare este:

```
sort -rn -k 2,2 file     ; sorteaza dupa a doua coloana
```

uniq

Se pot elimina duplicatele folosind optiunea **-u** la **sort**. Totusi, de multe ori este util sa afisam de cate ori apare un cuvant. Pentru aceasta vom folosi comanda **uniq** cu optiunea **-c**. Atentie! **uniq** face eliminarea duplicatelor numai daca liniile sunt sortate. Exemple de utilizare:

```
$ echo -en "alfa\nbeta\nbeta\nalfa\nbeta\ngamma\nalfa\n" | uniq
alfa
beta
alfa
beta
gamma
alfa
$ echo -en "alfa\nbeta\nbeta\nalfa\nbeta\ngamma\nalfa\n" | sort | uniq
alfa
beta
gamma
$ echo -en "alfa\nbeta\nbeta\nalfa\nbeta\ngamma\nalfa\n" | sort | uniq -c
      3 alfa
      3 beta
      1 gamma
```

wc

Comanda **wc** (word count) este folosita pentru a contoriza numarul de linii, de cuvinte sau de caractere dintr-un text sau de la intrarea standard. Pentru aceasta i se pot specifica optiunile **-c**, **-w**, **-l**.

cut

Comanda **cut** selecteaza numai anumite parti (coloane) ale fisierului de intrare sau ale intrarii standard. Sintaxa cea mai folosita este:

```
cut -d delim -f fields
```

Folosindu-se delimitatorul **delim** se vor selecta numai campurile **fields**. Exemple de utilizare sunt:

```
razvan@ragnarok:~/cfiles/solab/labs$ ls -l
total 44
drwxr-xr-x  2 razvan razvan 4096 Nov 16 00:01 lab1
drwxr-xr-x  3 razvan razvan 4096 Oct 12 00:02 lab2
-rwxr-xr-x  1 razvan razvan  937 Oct 19 00:53 lab2html.sh
drwxr-xr-x  3 razvan razvan 4096 Oct 19 00:50 lab3
drwxr-xr-x  3 razvan razvan 4096 Oct 26 01:09 lab4
drwxr-xr-x  3 razvan razvan 4096 Nov  2 09:47 lab5
drwxr-xr-x  3 razvan razvan 4096 Nov 12 19:56 lab6
drwxr-xr-x  2 razvan razvan 4096 Nov 16 12:43 lab7
drwxr-xr-x  2 razvan razvan 4096 Nov 20 14:38 lab8
drwxr-xr-x  5 razvan razvan 4096 Oct 17 19:52 solab
drwxr-xr-x  2 razvan razvan 4096 Nov 12 20:06 tema1
razvan@ragnarok:~/cfiles/solab/labs$ ls -l | cut -d ' ' -f 1,9
total
drwxr-xr-x 00:01
drwxr-xr-x 00:02
```

```
-rwxr-xr-x 19
drwxr-xr-x 00:50
drwxr-xr-x 01:09
drwxr-xr-x 2
drwxr-xr-x 19:56
drwxr-xr-x 12:43
drwxr-xr-x 14:38
drwxr-xr-x 19:52
drwxr-xr-x 20:06
razvan@ragnarok:~/cfiles/solab/labs$ ls -l | tr -s ' ' | cut -d ' ' -f 1,9
total
drwxr-xr-x lab1
drwxr-xr-x lab2
-rwxr-xr-x lab2html.sh
drwxr-xr-x lab3
drwxr-xr-x lab4
drwxr-xr-x lab5
drwxr-xr-x lab6
drwxr-xr-x lab7
drwxr-xr-x lab8
drwxr-xr-x solab
drwxr-xr-x tema1
```

Comenzi din aceeasi categorie sunt **paste** si **join**.

Exemple

In continuare sunt prezentate cateva exemple de script-uri shell.

```
#!/bin/bash

#
# Afisarea numelui de utilizator si a home directory-ului pentru
# utilizatorii care se afla in ultimele N login-uri in sistem, unde
# N este primit ca parametru
#
# Informatiile dorite sunt furnizate de /etc/passwd si de last -20
#

function usage ()
{
        echo "Usage: $0 num_login"
        exit 1
}

if test $# -ne 1; then
        usage
fi

# se obtine lista utilizatorilor
users=$(last -$1 | head -$(($1-2)) | cut -d ' ' -f 1 | sort | uniq | tr '\n' ' ')

for i in $users; do
        home_dir=$(cat /etc/passwd | grep "$i" | cut -d ':' -f 6)

        # no home dir
        if test -z $home_dir; then
                echo "User: $i Home dir: NONE"
```

```
        else
                echo "User: $i Home dir: $home_dir"
        fi
done

exit 0

#!/bin/bash

#
# Sa se mute copieze in directorul bkup toate fisierele modificate
# cu mai mult de 30 de zile in urma pentru utilizatorul curent
#

if ! test -d ~/bkup; then
        mkdir ~/bkup
fi

# cauta fisirele dorite
# do not try this if u have a lot of old files and little disk space ;)
find ~ -type f -ctime +30 -exec cp -u {} ~/bkup \;

exit 0
```

Exerciții

1. Creați şi rulați un script shell care afişează mesajul `Hello, World!`.

Hint:

- folosiți apostrof (`'`) pentru citare (`!` este văzut ca un caracter special)

2. Creați un script care afişează numărul de argumente primite. Dacă scriptul primeşte mai mult de două argumente atunci se afişează toate mai puțin primele două.

Hint:

- folosiți `shift`

3. Prelucrați scriptul anterior ca să afişeze atât argumentul cât şi numărul de caractere eventual cu un cap de tabel. Ceva de forma:

```
argument        valoare          dimensiune
3               alfa                    4
4               epsilon                 7
5               omega                   5
```

Hint:

- trebuie să folosiți `for`
- folosiți construcția `${#myvar}` pentru a afla numărul de caractere al variabilei `myvar`.

4. Creați un script care să primească un singur argument (scriptul se va întoarce cu eroare - `exit 1` - şi cu un mesaj specific dacă primeşte mai mult de un argument sau niciunul). Scriptul va analiza argumentul şi va afişa

mesajele:

- "argumentul începe cu o cifră"
- "argumentul începe cu o literă"
- "argumentul este o şir de caractere" pentru orice altă situație

Hint:

- folosiți `case`
- folosiți expresii regulate

5. Afişati numele utilizatorilor din sistem care încep cu litera 's'.

Hint:

- nu e nevoie de script shell; se poate face cu un one-liner
- folositi `/etc/passwd`, `grep` şi `cut`

6. Afişați separate prin tab numele şi shell-ul utilizatorilor din sistem care NU au home-ul în `/home/...`

Hint:

- nu e nevoie de script shell
- folosiți `/etc/passwd`, `grep`, `cut`, `tr`

7. Scrieți un script shell care să afişeze, separate prin virgula, directoarele in care se gasesc executabile `gcc`.

Hint:

- folositi `whereis gcc`, `basename` şi `dirname`

8. Creați un script shell care să afişeze suma memoriei ocupate de procesele din sistem după cum reiese din ieşirea comenzii `ps -e -o rss`. Comparați cu rezultatul oferit prin rularea comenzii `free`.

Hint:

- folosiți opțiuni `tail` pentru a afişa toate liniile mai puțin prima (prima este antetul `RSS`) (`man tail`)
- la tail este utila aici forma cu + a argumentului; daca nu gasiti in man nimic despre asta `info tail`
- folosiți expandare aritmetica

9. Un fişier conține linii de forma `nume parola`. Creați un script care să afişeze perechea `nume parola` numai dacă numele are 6 litere. Scriptul va primi fişierul ca argument.

Hint:

- folosiți `while read`

10. Creați un script care foloseşte un fişier în forma de mai sus. Scriptul trebuie să adauge în sistem utilizatorii. Parola trebuie schimbată la cea prezentă în fişier. Scriptul va primi ca argument numele fişierului de mai sus. Scriptul va fi non-interactiv - nu va cere utilizatorului nici o informație.

Hint:

- folosiți `useradd` (nu adduser - e interactiv) - verificați pagina de manual pentru opțiuni
- folosiți `chpasswd` pentru schimbarea în mod non-interactiv a parolei (`passwd` o schimbă neinteractiv)
- scriptul trebuie rulat ca root

11. Instalați serverul Apache2 (`apt-get install apache2`). Folosiți un script shell pentru a dezinstala toate pachetele instalate.

Hint:

- folosiți ieşirea comenzii `dpkg -l 'apache*'`; rândurile conținând pachetele instalate încep cu linia 'ii'
- pachetele pot fi dezinstalate prin transmiterea lor ca listă de argumente comenzii `apt-get remove --purge`
- folosiți opțiunea `-y` pentru a dezactiva interactivitatea `apt-get`

12. Dezinstalați pachetele de mai sus printr-un one-liner (o singură linie shell - fără nevoia unui script).

13. Folosiți un one-liner pentru a contoriza toate fişierele cu extensia `.sh` din ierarhia `/etc`.

Hint:

- folosiți `find`

14. Creați un script shell pentru a număra toate fişierele din `/etc` pentru care comanda `file` spune că sunt scripturi Shell:

```
razvan@asgard:~/junk/so$ file test.bash
test.bash: Bourne shell script text executable
```

Puteți să faceți scriptul un one-liner?

Hint:

- va trebui să folosiți `find` în combinație cu `for` (nu am găsit să se poată fără `for` ... mai caut)

15. Descărcați folosind `wget` arhiva cu sursele nucleului Linux 1.0 (ftp://ftp.eu.kernel.org/pub/linux/kernel/v1.0/linux-1.0.tar.bz2). Dezarhivați (`tar xjf ...`). Creați un script shell care să contorizeze numărul de linii din fişierele sursă.

Hint:

- fişierele sursă se consideră cele cu extensia `.c` sau `.h`

Fiecare exercițiu se punctează cu 1p (oricât de uşor/greu ar fi)

Link-uri utile

- http://www.linuxcommand.org/index.php
- http://www.comptechdoc.org/os/linux/programming/script/linux_pgscript.html
- http://tldp.org/LDP/abs/html/index.html
- http://wiki.lug.ro/mediawiki/index.php/Tutorial_Shell_Scripting
- http://www.gnu.org/software/bash/manual/bash.html (disponibil si in format info - **$ info bash**)